**CAPSTONE PROJECT- AUTOMATION TESTING ON DEMOBLAZE**

**Problem Statement**:Demoblaze is an online shopping platform where users can browse and purchase various electronic products.Users can browse through different categories and Users interact with the platform to view product details,add products to the cart,place orders.Automating these workflows is essential to ensure the platforms’s efficiency,accuracy,and reliability in the testing process.

**Objective**:The objective of this project is to automate the user workflows of the Demoblaze online shopping platform using Selenium,TestNG,Page Object Model and Page Factory and BDD Cucumber.This includes automating tasks such as viewing product details,adding products to the cart,placing orders.The aim is to enhance the robustness of the testing process,reduce manual effort,and ensure a seamless user experience.

**Introduction**:The Capstone Project is an automated testing framework built using Selenium,BDD Cucumber and TestNG. It follows the Page Object Model (POM) to structure test scripts efficiently. The project automates the testing of the “demoblaze” web application, handling functionalities like login, sign-up, cart operations, and reporting.

**TestFlow:**

SIGNUP

|

LOGIN

|

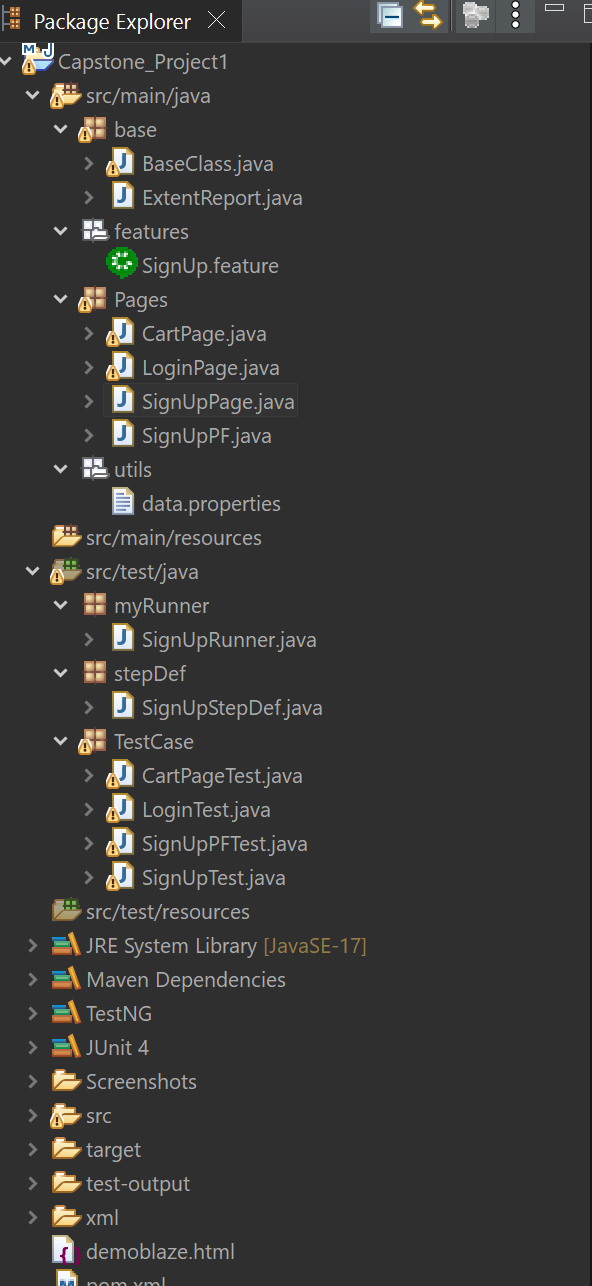
Add To Cart

**Key Features:**

* Page Object Model (POM) for maintainability.
* Cucumber BDD Framework for test case execution.
* TestNG for test execution and reporting.
* Selenium WebDriver for browser automation.
* Extent Reports for test reporting.
* Maven for dependency management.

1. **Project Structure**

Project url:https://www.demoblaze.com/



**2.** **Class Summaries**

**base (Package):**

**1. BaseClass.java**

* Initializes **WebDriver** and browser configuration by using @parameter.
* Manages common setup and teardown functions.

**2. ExtentReport.java**

* Generates **automated test reports** using Extent Reports.
* Captures logs and screenshots for failed test cases.

**features (Cucumber Feature File):**

**1.SignUp.feature:**

**SignUp.feature**

* A **Cucumber feature file** that defines **signup scenarios** in Gherkin syntax.
* Includes test cases like **successful signup** and **invalid input validation**.

**Pages** (Page Object Model )- POM Classes:

**1. SignUpPage.java**

* Handles user **sign-up functionality**.
* Contains methods to enter a username, password, and submit the form.
* Implements **element locators** using By.id() and By.xpath().

**2. LoginPage.java**

* Implements **user authentication methods**.
* Interacts with **login page elements** such as input fields and buttons.
* Validates **successful and failed login attempts**.

**3. CartPage.java**

* Automates **shopping cart operations**.
* Handles **adding and removing products** from the cart.
* Verifies **cart contents and checkout functionality**

**4. SignUpPF.java**

* Implements **Page Factory Model** for sign-up functionality.
* Uses @FindBy annotations for better **element handling**.
* Improves **test script efficiency and maintainability**.

**utils (Utilities):**

* data.properties(file) - Stores application URLs, user credentials, and configurations.

**myRunner (Cucumber Test Runner):**

**1. SignUpRunner.java**

* Executes Cucumber feature files using @CucumberOptions.
* Generates Cucumber HTML and JSON reports.

**stepDef (Step Definitions for Cucumber):**

**1.SignUpStepDef.java**

* Contains step definitions for Cucumber test scenarios.
* Maps Gherkin steps to Java methods.
* Uses TestNG assertions to validate results.

**TestCase (Test Scripts - TestNG Based):**

**1.CartPageTest.java:**

* Adds an item to the cart.
* Validates if the item appears in the cart.

**2.LoginTest.java:**

* Logs in with a valid username and password.
* Validates login success.

**3.SignUpPFTest.java:**

* Tests Page Factory-based for signup functionality.

**4. SignUpTest.java**

* A **TestNG test class** for validating sign-up functionality.
* Calls methods from **SignUpPage.java** for test execution.

**3. Other Important Files and Folders**

* src/test/resources - Stores test data, configurations.
* JRE System Library [JavaSE-17] - Java version used.
* Maven Dependencies - Required dependencies (Selenium, Cucumber, TestNG, Extent Reports).
* TestNG & JUnit 4 – Testing & Cucumber frameworks.
* Screenshots Folder - Stores screenshots of test cases.
* test-output Folder - Contains TestNG reports.
* xml Folder - May store TestNG XML configuration files.
* demoblaze.html - Local HTML file (results of extent reports).
* pom.xml - Maven configuration file managing dependencies.

**4. Execution Instructions**

**Run Cucumber Tests**

1. **Run from Eclipse:**

* Right-click SignUpRunner.java → Run As → JUnit Test.

**Run TestNG Tests**

1. **Run from Eclipse:**

* Right-click SignUpTest🡪Run AS 🡪TestNg Suite

Open test-output/index.html to check the **TestNG Reports**.

**5.Challenges Faced**

**1.Web Element Synchronization**

* + Handling **dynamic elements** and waiting strategies (Explicit & Implicit Waits).
  + Managing page load times and asynchronous operations.

**2.Cross-Browser Compatibility**

* + Ensuring tests run consistently across **Chrome, Firefox, and other browsers**.
  + Managing browser-specific behaviors and driver compatibility issues.

**3.Exception Handling**

* Managing **unexpected pop-ups, timeouts, and element not found exceptions**.
* Implementing robust try-catch blocks for smooth test execution.

**4.Data-Driven Testing**

* Implementing **parameterized tests using external data sources** (e.g., properties files).
* Managing test data efficiently for **multiple test scenarios**.

**5.Integration & Reporting**

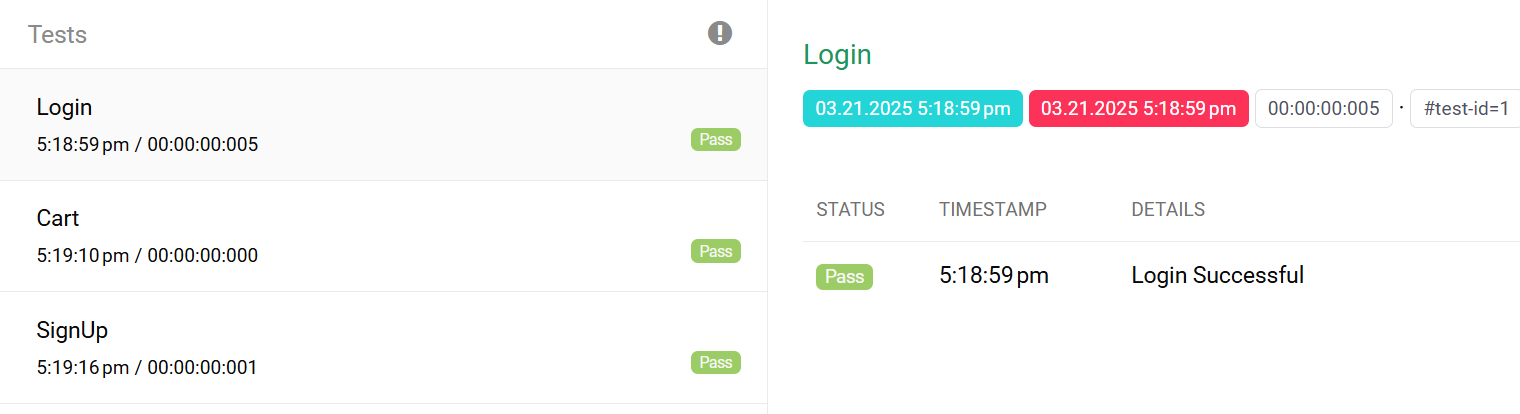
* Generating **Cucumber and Extent Reports** with clear and detailed logs.
* Debugging test failures based on captured **screenshots and logs**.

**6.Scalability & Maintainability**

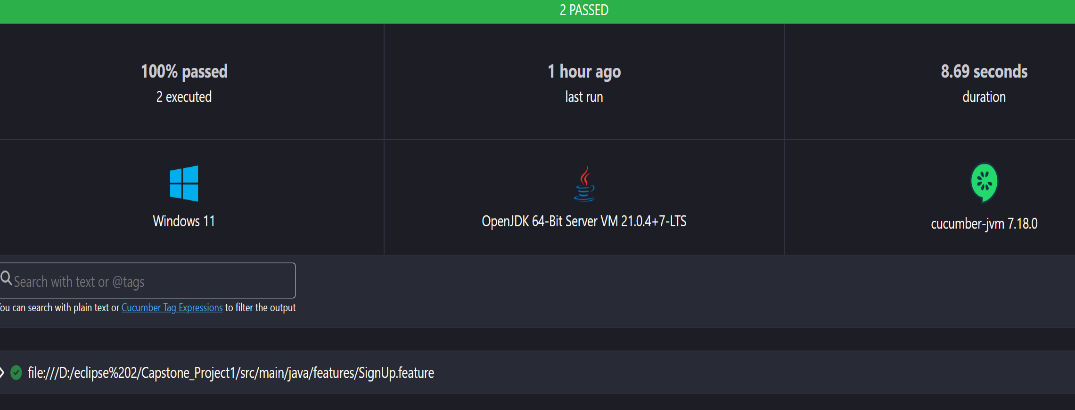
* Structuring the framework to support **future test cases** with minimal code changes.
* Managing large test suites efficiently with modularized code.

**6. Reports & Logs**

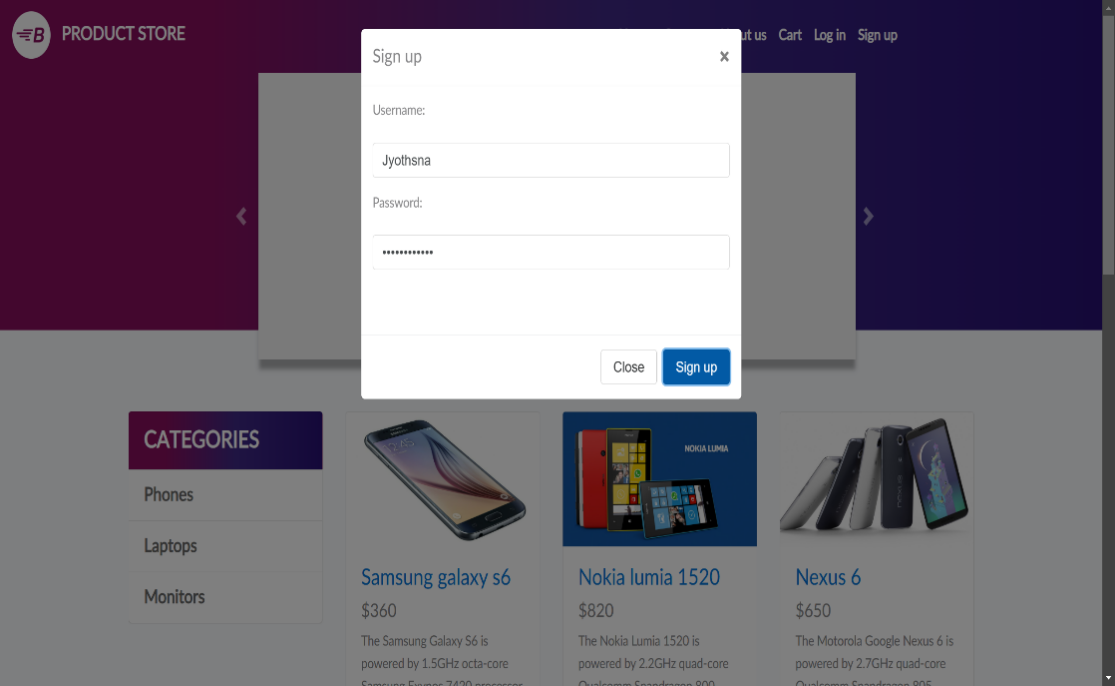
* **Extent Reports**: Generates detailed reports after execution (test-output/ExtentReports.html).



* **Cucumber Reports**: Located in target/cucumber-reports/index.html.



* **Screenshots**: Captured inside Screenshots/ folder.



* **TestNG XML Files**: Located in xml/ directory and used for configuring test execution (Login.xml, SignUppf.xml).
* <?xml version="1.0" encoding="UTF-8"?>
* <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
* <suite name="Suite">
* <test thread-count="5" name="Test">
* <parameter name = "browser" value = "chrome"/>
* <classes>
* <class name="TestCase.LoginTest"/>
* <class name="TestCase.CartPageTest"/>
* <class name = "TestCase.SignUpTest"/>
* </classes>
* </test> <!-- Test -->
* </suite> <!-- Suite -->
* **Test Output Folder**: Contains logs, execution details, and generated reports from TestNG.

**7. Conclusion**

This Selenium project follows **Page Object Model (POM) + Cucumber (BDD) + TestNG** for structured automation testing. The project ensures **reusability, maintainability, and scalability**.